
# Funktionaalinen ohjelmointiparadigma web-käyttöliittymäkehityksessä

Oulun yliopisto
Tietojenkäsittelytieteiden tutkinto-ohjelma
LuK-tutkielma
Antti Pilto
7.5.2017

# Tiivistelmä

Nykyaikaisten web-sovellusten lisääntynyt interaktiivisuus ja uudenlaiset teknologiat – kuten mobiili- ja muut älylaitteet - ovat lisänneet web-käyttöliittymiin kohdistuvia vaatimuksia, mikä on lisännyt tarvetta kehittää uusia näkökulmia, lähestymistapoja ja teknologioita web-käyttöliittymäkehitykseen. Funktionaalinen ohjelmointi on lambdakalkyyliin perustuva ohjelmointiparadigma. Paradigmalla on pitkä historia, ja siitä on tehty lukuisia tieteellisiä tutkimuksia. Yleinen vertailukohde on imperatiivinen ohjelmointiparadigma, johon suurin eroavaisuus löytyy sivuvaikutuksien puutteesta ja siitä seuraavista eduista ja haitoista. Tässä tutkielmassa tarkastellaan kirjallisuuskatsauksen keinoin, mitä funktionaalisen ohjelmointiparadigman tärkeimpiä ominaisuuksia voidaan hyödyntää web-käyttöliittymäkehityksessä, ja millä tavalla niitä voidaan hyödyntää. Vastauksena tutkimuskysymykseen tutkielma osoittaa kaikkien funktionaalisen ohjelmoinnin tärkeimpien ominaisuuksien hyödyntämisen olevan web-käyttöliittymäkehityksessä mahdollista, mutta osoittaa myös tarpeen JavaScript-kirjastojen tai vaihtoehtoisten ohjelmointikielten käytölle. Tutkielma esittää myös tarjottavien ratkaisujen seuraukset, ja tarpeen uusien teknologioiden ja esitettyjen ratkaisujen soveltuvuuden lisätutkimukselle.

*Avainsanat*
Funktionaalinen ohjelmointi, web-ohjelmointi, JavaScript

*Ohjaaja*
Yliopisto-opettaja, Henrik Hedberg

# Sisällys

# 1. Johdanto

Web on historiansa aikana kehittynyt nopeatempoisesti, ja sen alkuperäisen käyttötarkoituksen muuttuminen on johtanut uusien teknologioiden kehittymiseen, ja kehitysprosessin muuttumiseen. Lisääntynyt web-sovellusten interaktiivisuus ja uudenlaiset teknologiat – kuten mobiililaitteet - ovat lisänneet käyttöliittymiin kohdistuvia vaatimuksia. Esimerkiksi Fedosejevin (2015) kuvaamat yhden sivun sovellukset (Single Page Application, SPA) ja Robbinsin (2012) kuvaama responsiivinen suunnittelu ovat nykyaikaisessa web-sovelluskehityksessä merkittävässä osassa.

Ohjelmistojen kompleksisuuden kasvaessa perinteisissä ohjelmointiparadigmoissa, kuten proseduraalisessa ja olio-ohjelmoinnissa, voidaan havaita ongelmia, kuten ohjelmasuorituksen korkeat resurssivaatimukset, ja funktioiden sivuvaikutuksien aiheuttamat ongelmat. Useiden tutkimuksien ja kirjallisuuden mukaan (esim. Hughes, 1989; Hu, Hughes & Wang, 2015; Vesterholm & Kyppö, 2015) matemaattisiin funktioihin pohjautuva funktionaalinen ohjelmointiparadigma kykenee ratkaisemaan näitä ongelmia sivuvaikutuksien puuttumisella ja sen mahdollistamalla rinnakkaissuorituksella.

Koska nykyaikaiset web-sovellukset ovat yhä useammin laajoja ja monimutkaisia ohjelmistoja, on aiheellista esittää tutkimuskysymys, mitä funktionaalisen ohjelmointiparadigman tärkeimpiä ominaisuuksia voidaan hyödyntää web-käyttöliittymäkehityksessä ja millä tavalla niitä voidaan hyödyntää. Tässä tutkielmassa tähän tutkimuskysymykseen vastataan kirjallisuuskatsauksen keinoin.

Web-sovellusten kehitys voidaan pääsääntöisesti jakaa selainpuolen (frontend) ja palvelinpuolen (backend) kehitykseksi (Robbins, 2012). Tutkimusaiheen rajaamiseksi tässä tutkielmassa keskitytään selainpuolen käyttöliittymäkehitykseen ja sen teknologioihin.

Tutkielman alussa perehdytään web-käyttöliittymäkehitykseen ja funktionaalisen ohjelmointiparadigman ominaisuuksiin. Nämä kaksi kokonaisuutta pyritään yhdistämään analysoimalla funktionaaliseen ajattelutapaan ja web-käyttöliittymäkehitykseen soveltuvia ohjelmointiteknologioita. Tutkielma päätetään löydöksiin perustuvalla pohdinnalla ja yhteenvedolla.

# 2. Web-käyttöliittymäkehityksen vaatimukset ja erityispiirteet

Web on nyt ollut olemassa yli 20 vuotta, ja sen kehityshistoria on ollut jatkuvaa evoluutiota. Varmaa on, että web kommunikointi- ja kaupallisena välineenä ei tule katoamaan. Web on perinteisten tietokoneiden lisäksi laajentunut muihin laitteisiin, kuten älypuhelimiin, tabletteihin ja televisioihin. (Robbins, 2012)

## 2.1 Web-teknologiat

Web-kehityksessä käytettävät yleisimmät teknologiat ovat Hypertext Markup Language (HTML), Cascading Style Sheets (CSS), JavaScript, sekä palvelinpuolen ohjelmointi ja tietokannan hallinta. Web-teknologioiden kehittymistä valvoo ja ohjaa vuonna 1994 perustettu World Wide Web Consortium (W3C). (Robbins, 2012)

Web-sovellukset ovat web-teknologioihin pohjautuvia verkkosovelluksia. Web-sovellus voi olla yksinkertainen selainkeskeinen web-sivusto tai kehittynyt ohjelmisto, kuten esimerkiksi Google Docs. Yhteydenpito selaimen ja palvelimen välillä tapahtuu HTTP-protokollalla. (Cervesato & Sansyz, 2014) Web-sovelluksen etuna perinteisiin käyttöjärjestelmälle natiiveihin sovelluksiin verrattuna on käyttäjän kannalta helppo käyttöönotto - asennusohjelmia ei tarvita koska kaikki tapahtuu web-selaimessa. (Cervesato & Sansyz, 2014; Richard-Foy, Barais & Jezequel, 2014)

Nykyisin useat web-sovellukset eivät koostu kokoelmasta yksittäisiä web-sivuja. Sen sijaan on alettu kehittää web-sovelluksia, jotka koostuvat vain yhdestä sivusta, joka ei kuvaa sisällön rakennetta, vaan säiliötä johon web-sovellus sijoitetaan. Tällaisia web-sovelluksia kutsutaan yhden sivun sovelluksiksi (Single Page Application, SPA). (Fedosejev, 2015)

Web-sovellusten kehitys on monella tapaa erilaista ja monimutkaisempaa kuin perinteinen ohjelmistokehitys. Web-sovellukseen kohdistuvilla vaatimuksilla on tapana kasvaa nopeasti ja niiden sisältö muuttuu jatkuvasti. (Ginige & Murugesan, 2001) Lisäksi siihen kohdistuu useita ohjelmistoteknisiä erityisvaatimuksia. Käyttäjien määrä eri ajankohtina vaihtelee merkittävästi, joten sovelluksen täytyy sietää raskasta kuormitusta, ja latautumisaikojen tulisi aina pysyä pienenä. Myös käyttöliittymän esteettisyys, tietoturva ja sisällön laatu ovat merkittävässä asemassa. (Pressman, 2010)

Web-kehitys voidaan pääsääntöisesti jakaa selainpuolen (frontend) ja palvelinpuolen (backend) kehitykseksi. Selainpuolen kehitys koostuu niistä osa-alueista, jotka näkyvät käyttäjän selaimessa, kuten graafinen- ja käyttöliittymäsuunnittelu, HTML-rakenne ja CSS-tyyliohjeet, ja JavaScript-toiminnallisuus. Palvelinpuoli koostuu niistä taustalla tapahtuvista toiminnoista, jotka mahdollistavat web-sovelluksen dynaamisen ja interaktiivisen toiminnan. Palvelinpuolen kehityksen osa-alueita ovat esimerkiksi lomakkeiden prosessointi, tietokantaohjelmointi ja sisällönhallintajärjestelmät. (Robbins, 2012)

HTML on kieli, jolla luodaan web-dokumentteja. Nykyisin HTML 4.01 ja HTML5 ovat yleisimmät käytössä olevat HTML-versiot. (Robbins, 2012) Marraskuussa 2016 W3C julkaisi suositukset HTML 5.1-versiolle (World Wide Web Consortium, 2016). HTML ei ole ohjelmointikieli vaan kuvauskieli. Sillä tunnistetaan ja kuvataan dokumentin

erilaisia komponentteja, kuten otsikkoja, kappaleita ja listoja. Se kuvaa siis sitä rakennetta, josta dokumentti koostuu. (Robbins, 2012)

CSS on kieli, jolla määritellään web-sivun sisällön esitystapa, kuten värit, asettelu ja fontit. Viimeisin CSS:n versio on CSS3. (World Wide Web Consortium, 2016) HTML:n kuvatessa web-sivun sisältöä, CSS määrittelee, miltä kyseinen sisältö näyttää (Robbins, 2012). CSS on itsenäinen HTML:stä ja sitä voidaan käyttää minkä tahansa XML-pohjaisen kuvauskielen kanssa. HTML:n ja CSS:n erottaminen tekee helpommaksi web-järjestelmän ylläpidon, tyyliohjeiden jakamisen eri sivujen välillä, ja sivustojen räätälöimisen eri ympäristöjen mukaiseksi. Tätä erottamista nimitetään rakenteen ja esitystavan erottamiseksi. (World Wide Web Consortium, 2016)

CSS mahdollistaa sisällön esitystavan muuntautuvuuden erilaisten laitteiden, kuten suurten ja pienten näyttöjen tai tulostimen mukaan (World Wide Web Consortium, 2016). Web-sivuun kohdistetaan siis erilaisia tyylimäärityksiä näytön koon mukaan. Tätä ominaisuutta kutsutaan responsiivisuudeksi (Robbins, 2012).

Responsiivisella suunnittelulla (responsive design) voidaan luoda sivustolle esimerkiksi täysin erilainen käyttöliittymä, kun sitä käytetään mobiiliselaimella. Responsiivisuus mahdollistaa tyydyttävän käyttökokemuksen säilymisen erilaisten käytettävien laitteiden ja resoluutioiden välillä. (Robbins, 2012) CSS3:lla toteutus tapahtuu mediakyselyjen (media query) avulla (World Wide Web Consortium, 2016).

## 2.2 JavaScript

Koska web on kehittynyt merkittäväksi sovellusalustaksi, ohjelmoinnin osuus web-kehityksessä on tullut tärkeäksi. JavaScriptistä on tullut olennainen osa web-kehitystä. (Robbins, 2012)

JavaScript on kevyt, tulkattava ohjelmointikieli, jonka avulla web-sivustojen ei tarvitse olla pelkkää staattista HTML-sisältöä, vaan ne voivat sisältää ajettavia ohjelmia, jotka lisäävät interaktiivisuutta eli vuorovaikutusta käyttäjän kanssa, hallitsevat selainta, ja luovat dynaamista HTML-sisältöä (Flanagan, 2006, Robbins, 2012). Esimerkkejä interaktiivisuudesta ovat reaaliaikainen palautteen antaminen käyttäjälle sivustolla navigoitaessa, lomakkeiden tarkistus, ja “lennossa” luotavat käyttäjän valintojen mukaiset sivut (Negrino & Smith, 2007).

JavaScriptin syntaksi muistuttaa pohjimmiltaan C-, C++- ja Java-kieliä. Kielessä käytetään tyypillisiä rakenteita, kuten *if*, *while*, ja *&&*. Kielten samankaltaisuudet suurilta osin kuitenkin rajoittuvat syntaksiin. (Flanagan, 2006)

JavaScript on tyypitön kieli, toisin sanoen muuttujilla ei ole määriteltyä tietotyyppiä (Flanagan, 2006). Sitä voidaan myös kutsua heikosti tai dynaamisesti tyypitetyksi kieleksi, jotka viittaavat samaan ominaisuuteen (Cervesato & Sansyz, 2014).

Ensimmäinen JavaScript-versio julkaistiin Netscape 2-selaimessa (Flanagan, 2006). JavaScriptin alkutaipaleella eri selaintoimittajat kulkivat omiin suuntiinsa kielen kehityksen kanssa. Siksi kansainvälinen standardointiorganisaatio ECMA julkaisi vuonna 1997 Netscapen ja Microsoftin avustuksella ECMAScript-standardin. Nykyisin kaikki selainvalmistajat huolehtivat selaimensa JavaScript-toteutuksen ECMAScript-yhteensopivuudesta. Viimeisin tuettu ECMAScript-versio vaihtelee selainten välillä. (Negrino & Smith, 2007)

### 2.2.1 Rajoitukset

JavaScript on alun perin suunniteltu toimimaan selaimessa käyttäjän tietokoneella, ei palvelimella. Tämä johtaa tiettyihin rajoituksiin. Asiakaskoneella selaimen ajama JavaScript ei voi lukea tai kirjoittaa kovalevyllä sijaitsevia tiedostoja, pois lukien evästeet. Myöskään palvelimelle ei voida suoraan luoda tiedostoja, tähän vaaditaan yhteys palvelimella sijaitsevaan ohjelmaan. JavaScript kykenee hallitsemaan vain luomiaan ikkunoita, joten muihin avoinna oleviin välilehtiin ja ikkunoihin ei pystytä puuttumaan. (Negrino & Smith, 2007)

JavaScript on kuitenkin yleiskäyttöinen kieli, josta johtuen nykyisin on mahdollista käyttää JavaScriptiä muissakin kuin selainympäristöissä, kuten palvelinpuolen ohjelmointikielenä, esimerkiksi Node.js ympäristössä (Flanagan, 2006; Richard-Foy ym., 2011).

### 2.2.2 DOM-manipulaatio

HTML-sivu voidaan kuvata sen oliomallin mukaisena puurakenteena. Tätä esitystapaa kutsutaan dokumentin oliomalliksi (Document Object Model, DOM). JavaScript voi hallita DOM-puuta poistamalla, lisäämällä, muuttamalla tai tutkimalla sen solmuja. Solmujen manipulointi johtaakin oleelliseen eroon dynaamisen web-ohjelmiston ja staattisen HTML-sivun välillä. (Negrino & Smith, 2007)

DOM-manipulaation avulla on mahdollista luoda kokonainen dokumentti tyhjästä. DOM:in ominaisuuksia manipuloimalla voidaan määritellä dokumentin tai sen elementin käyttämät tyylit ja sisällön. Tällainen dynaaminen sisällön luominen mahdollistaa joissain tapauksissa palvelimella tapahtuvan ohjelmasuorituksen korvaamisen kokonaan selaimessa ajettavalla JavaScriptillä. (Flanagan, 2006)

### 2.2.3 Tapahtumat

Käyttäjän web-sivulla suorittamat toiminnot aiheuttavat tapahtumia (event). Yleisiä esimerkkejä tapahtumista ovat onClick-tapahtuma käyttäjän napauttaessa hiiren painiketta, tai onMouseover-tapahtuma kursorin ollessa tietyn olion päällä. Näitä tapahtumia JavaScript käsittelee tapahtumankäsittelijöillä (event handlers). (Negrino & Smith, 2007)

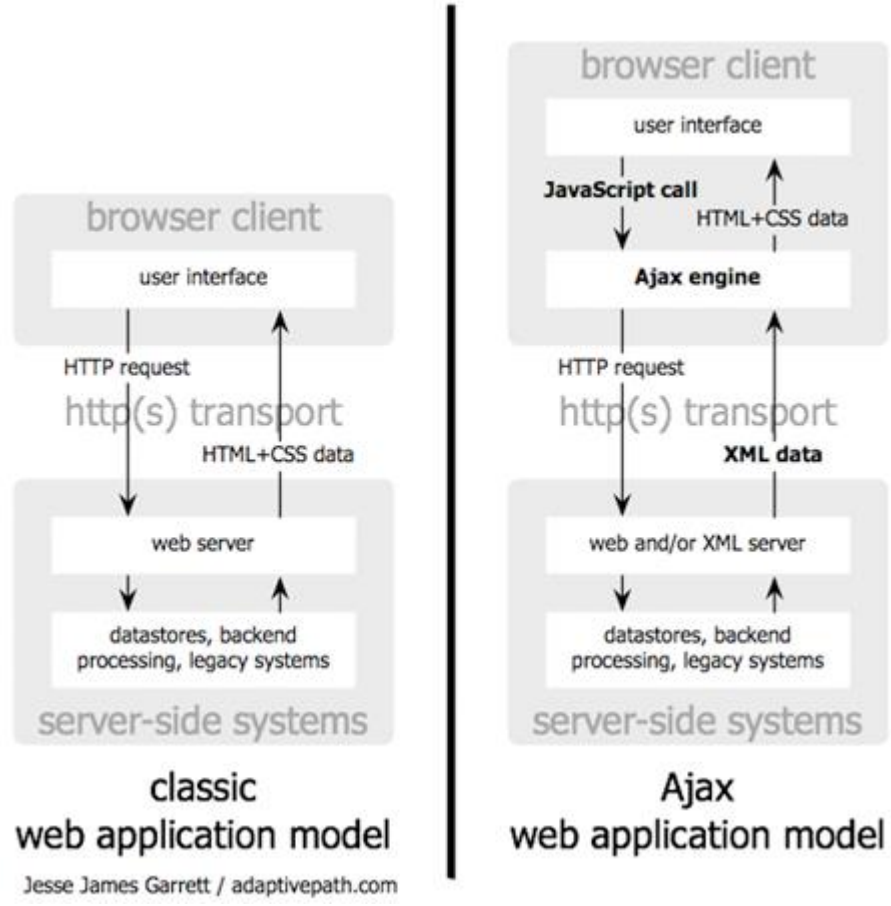
Tapahtumankäsittelijät mahdollistavat vuorovaikutuksen käyttäjän ja ohjelman välillä. Esimerkiksi onSubmit-tapahtuma voidaan määritellä käynnistämään syötteentarkistus-ohjelman lomakkeen lähetyksen yhteydessä. (Flanagan, 2006)

### 2.2.4 Asynkroniset HTTP-pyynnöt

Web-pohjaisten järjestelmien klassinen tapa toimia on ollut, että jokainen käyttäjän suorittama toiminto aiheuttaa HTTP-pyynnön palvelimelle. Palvelin ajaa pyyntöön pohjautuvia ohjelmasuorituksia ja palauttaa käyttäjälle uuden HTML-sivun. Malli perustuu alkuperäiseen webin käyttötarkoitukseen, hypertekstin tarjoamiseen. Vaikka malli on teknisesti järkevä, se ei kuitenkaan sovellu hyvin nykyaikaisiin web-sovelluksiin. (Garrett, 2005)

Ajax (Asynchronous JavaScript and XML) on joukkoa web-teknologioita hyödyntävä menetelmä, joka vähentää nykivää interaktiota web-pohjaisissa järjestelmissä lisäämällä käyttäjän ja palvelimen väliin yhden kerroksen (Kuva 1). Menetelmä siis mahdollistaa HTTP-pyyntöjen suorittamisen ja niihin reagoimisen ilman sivun uudelleen piirtämistä. (Garrett, 2005)

Käyttäjän suorittama toiminto joka tavallisesti aiheuttaisi HTTP-pyynnön, lähetetään sen sijaan JavaScriptin avulla Ajax-moottorille, joka palauttaa vastauksen taustalla suoritettavan HTTP-pyynnön avulla. Tästä johtuen käyttäjä ei joudu toiminnon suoritettuaan joudu odottamaan palvelimelta tulevaa vastausta, vaan voi jatkaa web-sovelluksen käyttöä. (Garrett, 2005)




**Kuva 1.** Vasemmalla perinteinen malli, oikealla Ajax-malli (Garrett, 2005).

## 2.3 Web-kehityksen ongelmat

Sovelluksen kasvaessa kehityksen monimutkaisuus kasvaa ja bugit ovat vaikeammin havaittavissa ja hahmotettavissa. Cervesato & Sansyz:n (2014) mukaan tähän vaikuttaa kaksi tekijää. Web-sovellukset vaativat hajautettua laskentaa, joka itsessään on vaikeaa. Web-sovellusten kehitys vaatii myös useiden heterogeenisten ohjelmointi-, skriptaus-, ja kuvauskielten hallintaa. (Cervesato & Sansyz, 2014) Dokumentin rakenteessa käytetään HTML:ää, dokumentin muotoiluun CSS:ää, interaktiivisuuden luomisessa (DOM-manipulaatio) JavaScriptiä, tiedon välitykseen selaimen ja palvelimen välillä HTTP protokollaa ja REST-rajapinnan tai SOAP-protokollan avulla välitettävää JSON- tai XML-tietoformaattia, SQL-kieliä pysyvään tiedon tallentamiseen, ja kaiken lisäksi palvelinpuolen ohjelmointikieleksi on monia vaihtoehtoja, yleisimmin PHP, Java, Ruby

on Rails, .NET ja Python. (Cervesato & Sansyz, 2014; Chlipala, 2016) Kielten välistä kommunikaatiota vaikeuttaa muun muassa yhteensopivien ja staattisten tietotyyppien puuttuminen - esimerkiksi JavaScriptissä tyypitys on dynaamista (Cervesato & Sansyz, 2014).

JavaScriptin käytön loppuminen web-kehityksessä ei ole odotettavissa. On kuitenkin tarpeellista muistaa, että se ei ole täydellinen kieli. JavaScriptissä on paljon outoa toiminnallisuutta ja se ei ole ilmaisultaan kovinkaan ytimekäs. Yhtäkkinen kaikkien JavaScriptin ongelmien korjaaminen kuitenkin johtaisi ”webin rikkoutumiseen” koska kieli muuttuisi niin paljon, että olemassa olevat ohjelmistot eivät välttämättä voisi toimia uuden tulkin kanssa. (Fogus, 2013)

Vaikka kielellä on ongelmansa, tietyillä menetelmillä JavaScript-ohjelmakoodista saadaan turvallista, skaalautuvaa, ja helppoa ymmärtää ja testata. Yksi näistä menetelmistä on Fogusin (2013) mukaan funktionaalinen ohjelmointi.

Etenkin yhden sivun sovellukset vaativat suuren määrän DOM-manipulaatioita. Suora DOM:n manipuloiminen JavaScriptillä tuo mukanaan kaksi ongelmaa. Ohjelmointimenetelmä on imperatiivinen, joka johtaa mm. vaikeasti ylläpidettävään lähdekoodiin. DOM:n manipuloiminen on myös hidasta, koska sen nopeutta ei voida optimoida toisin kuin muuta JavaScript-koodia. (Fedosejev, 2015)

# 3. Funktionaalisen ohjelmoinnin ominaisuudet

Funktionaalinen ohjelmointiparadigma pohjautuu matemaattisiin funktioihin, joilla lopputulos lasketaan annettujen lähtötietojen avulla. Funktionaalinen puhtaus tarkoittaa, että palautettava tulos riippuu ainoastaan annetuista parametreista, joten kutsuttava funktio tuottaa samoilla parametreilla aina saman lopputuloksen, eikä se voi muuttaa järjestelmän tilaa (state) ajon aikana – tarkoittaen, että funktiolla ei voi olla minkäänlaisia sivuvaikutuksia (side effects). (Vesterholm & Kyppö, 2015; Hughes, 1989; Hu ym., 2015) Funktionaalisesti puhtaassa ohjelmoinnissa ei käytetä sijoituslauseita, eli muuttujan arvo ei voi muuttua arvon saatuaan (immutable object) (Hughes, 1989).

Imperatiivisessa ohjelmointiparadigmassa ohjelma määritellään ketjulla komentoja jotka palauttavat lopputuloksen, mutta samalla muokkaavat järjestelmän tilaa (Bakker & De Vink, 2001; Hu ym., 2015). Tyypillisesti tilat ovat muuttujien joukkoja, ja komennot sisältävät tilan muuttujiin kohdistuvia muokkauksia. Ohjelman koostuminen komentoketjuista johtaa tiettyyn ketjuun välitiloja (intermediate states). Välitilat ovat havainnoitavia (observable), eli välitiloja voidaan tarkastella esimerkiksi hakemalla sen sisältämän muuttujan arvon. (Bakker & De Vink, 2001)

Tilan muuttuminen imperatiivisessa ohjelmoinnissa on vain yksi mahdollisista sivuvaikutuksista - ohjelma voi aiheuttaa poikkeuksia, tai huonoimmassa tapauksessa muita odottamattomia ohjelmasuorituksia, esimerkiksi ”sähköpostin lähetyksen tai ohjuksen laukaisun” (Hu ym., 2015). Suurin osa ohjelmointikielistä sallivat sivuvaikutuksien tapahtumisen ilman mitään varoituksia (Hu ym., 2015). Sivuvaikutuksien puuttuminen funktionaalisesta ohjelmoinnista eliminoi yhden merkittävän ongelmien lähteen, ja helpottaa ohjelmiston virheenkorjausta (Hughes, 1989).

Koska sivuvaikutus ei voi muuttaa lausekkeen tulosta, funktionaalisuus vähentää ohjausvuon (control flow), eli ohjelman osien suoritusjärjestyksen merkitystä (Hughes, 1989; Knight, 1986). Mahdollisuus osien rinnakkaiseen suoritukseen on siis helpompaa havaita (Hudak & Bloss, 1985). Rinnakkainen suoritus hyödyntää moniytimisten prosessorien laskentatehoa tehokkaammin kuin peräkkäinen suoritus. Tällaista funktionaalisuuteen perustuvaa ratkaisua käytetään muissakin kuin funktionaalisesti puhtaissa kielissä, esimerkiksi Java-kielen Stream API hyödyntää funktionaalisia liittymiä rinnakkaisuuden toteuttamiseen. (Vesterholm & Kyppö, 2015)

## 3.1 Vertailu muihin ohjelmointiparadigmoihin

Hughes (1989) toteaa, että kielestä on mahdotonta tehdä tehokkaampaa pelkästään jättämällä siitä pois ominaisuuksia, kuten sijoituslauseet. Toisaalta Hudakin ja Blossin (1985) mukaan funktionaalisen ohjelmoinnin kannattajat näkevät sijoituslauseiden puutteen positiivisena ominaisuutena, koska se estää sivuvaikutukset, mikä johtaa rinnakkaisen suorituksen mahdollistamiseen. Imperatiivinen ohjelmoija kokee saman asian hidasteena. Esimerkiksi vektoreiden, matriisien ja moniulotteisten taulukoiden muokkaamiseksi niiden sisältämää tietoa joudutaan kopioimaan, joka on laskennallisesti kallista. (Hudak & Bloss, 1985) Usein on luonnollisempaa ajatella algoritmeja tilojen ja niiden siirtymien näkökulmasta. Tämä imperatiivinen ajattelutapa ei sovi funktionaalisen laskennan lähestymistapaan. Samat algoritmit voidaan toteuttaa molemmilla tavoilla, mutta aiemmin mainittu tapa on useiden ohjelmoijien mielestä helpompi ajatella ja suunnitella, ja se helpottaa virheenkorjausta. (Knight, 1986)

Yhtä funktionaalisen ohjelmoinnin eroa olio-ohjelmointiin Fogus (2013) kuvaa sillä, miten ongelmat ryhmitellään ja nimetään. Olio-ohjelmoinnissa ylimmällä kuvaustasolla ovat substantiivimuotoiset oliot (esim. Button, Panel, Client), funktionaalisessa ohjelmoinnissa verbimuotoiset funktiot, eli toiminnot (esim. toHTML, postProcess, modifyDOM). (Fogus, 2013)

Kuten aiemmin mainittiin, imperatiivisessa ohjelmoinnissa laskenta tapahtuu tilasiirtymien ketjuna. Funktionaalinen ohjelmointi on lausekkeita korostava, tilaton ohjelmointimenetelmä. Kuten matemaattisissa funktioissakin, funktionaalisessa ohjelmoinnissa keskitytään siihen mitä lasketaan, ei siihen, kuinka lasketaan. (Hu ym., 2015)

Hun ym. (2015) esimerkissä määritellään matemaattinen kertomafunktio:

0! = 1
(n + 1) = (n + 1)n!

Sama määriteltäisiin funktionaalisella Haskell-kielellä seuraavasti:

```
fac 0 = 1
fac (n + 1) = (n + 1) * fac n
```

Struktuuriltaan määritelmä siis pysyy samana. Imperatiivisella ohjelmoinnilla vastaava tulos saavutettaisiin tilan muutoksien ja sijoituslauseiden avulla esimerkiksi seuraavasti:

```
n = x;
s = 1;
while (n>0) do {
        s = s*n;
        n = n-1;
}
```

Esimerkistä voidaan huomata funktionaalisen ohjelmoinnin olevan usein selkeämpää ja lyhempää kuin imperatiivinen ohjelmointi. Tämä johtaa korkeampaan tuottavuuteen ja laadukkaampaan lopputulokseen. (Hu ym., 2015)

Suurin eroavaisuus imperatiivisiin menetelmiin kuitenkin on havaittavissa ’=’ symbolin käytössä. Imperatiivisessa ohjelmassa ’=’ päivittää destruktiivisesti sen oikealla puolella olevan arvon symbolin vasemmalla puolella olevaan muuttujaan. Funktionaalisessa ohjelmassa ’=’ symboloi aina pätevää yhtäläisyyttä; aiemmassa esimerkissä *fac* 0 on yhtä kuin 1 kontekstista riippumatta. Tämä ominaisuus, jota kutsutaan viittauksen läpinäkyvyydeksi (referential transparency) on osa funktionaalista puhtautta, ja vaikuttaa olennaisesti siihen, miten ohjelma rakennetaan. (Hu ym., 2015)

## 3.2 Struktuuri ja modulaarisuus

Ohjelmiston kompleksisuuden kasvaessa ohjelmakoodin struktuuri tulee tärkeämmäksi. Hyvin strukturoitu ohjelmisto on helppo kirjoittaa ja ylläpitää, ja se mahdollistaa helpomman virheenkorjauksen ja tehokkaamman ohjelmakoodin uudelleenkäytön. (Hughes, 1989)

Merkittävin ero strukturoidun ja strukturoimattoman ohjelman välillä on, että strukturoidut ohjelmat suunnitellaan modulaarisesti. Modulaarinen malli aiheuttaa merkittävää tuottavuuden kasvua. Pienet moduulit voidaan kehittää nopeasti ja helposti.

Yleiskäyttöisiä moduuleita voidaan uudelleenkäyttää, joka johtaa nopeampaan ohjelman laajennuksien kehittämiseen. Ohjelmamoduuleita voidaan testata yksikköinä, joka auttaa virheenkorjaukseen käytetyn ajan vähentämisessä. (Hughes, 1989)

Ratkaistaessa ongelmaa modulaarisesti, ongelma jaetaan ensin aliongelmiin, ratkaistaan sitten aliongelmat, ja lopulta yhdistetään ongelmien ratkaisut. Tavat jolla alkuperäinen ongelma voidaan jakaa, ovat riippuvaisia menetelmistä joilla ratkaisut voidaan yhdistää. Ongelman modularisointia voidaan Hughesin (1989) mukaan parantaa tuomalla ohjelmointikieleen uusia tapoja, “liimoja”, ratkaisujen yhdistämiseen. Funktionaalinen ohjelmointi tarjoaa ominaispiirteenään kahta tärkeää “liimaa”: korkeamman asteen (higher order) funktiot ja laiska evaluointi (lazy evaluation). (Hughes, 1989; Achten & Koopman, 2013; Hu ym., 2015)

### 3.2.1 Korkeamman asteen funktiot

Jos käytettävässä ohjelmointikielessä funktiolle voidaan määritellä nimi, siihen voidaan viitata määritellyllä nimellä, sitä voi käyttää parametrina, ja sitä voidaan käyttää osana lauseketta, voidaan sanoa funktioiden olevan ohjelmointikielessä ensiluokkaisia. Funktionaalisessa ohjelmointikielessä funktion ensiluokkaisuus on välttämätöntä, jotta sillä voidaan toteuttaa korkeamman asteen funktioita. (Backhouse, Jansson, Jeuring & Meertens, 1998)

Korkeamman asteen funktiolle voidaan antaa parametriksi funktioita ja sen paluuarvo voi olla funktio. Tämän seurauksena on mahdollista liittää yhteen yksinkertaisia funktioita jotka muodostavat monipuolisemman funktion. (Hughes, 1989) Laskennallisen ongelman ratkaisussa korkeamman asteen funktiot parantavat ratkaisun abstraktiotasoa koska yhden ongelman ratkaisun sijaan kehitetään ratkaisu kokonaiselle luokalle ongelmia (Achten & Koopman, 2013).

Mahdollisuus käyttää funktioita parametreinä ja paluuarvoina on yksi funktionaalisen ohjelmoinnin kulmakivistä (Vesterholm & Kyppö, 2015). Myös muut kuin funktionaalisesti puhtaat ohjelmointikielet, kuten suositut C#, C++ ja Java, ovat alkaneet tukea lambda lausekkeita, jotka mahdollistavat korkeamman asteen funktioiden käytön (Hu ym., 2015).

### 3.2.2 Tarvepohjainen suoritus

Tarvepohjainen suoritus (lazy evaluation) on seurausta aiemmin mainituista funktionaalisen ohjelmoinnin perusperiaatteista, viittauksen läpinäkyvyydestä ja suoritusjärjestyksen merkityksettömyydestä (Achten & Koopman, 2013; Wadler, 1995). Esimerkkitapauksessa

$$(g \cdot f)$$

ohjelma f ajetaan vain ohjelman g tarvitessa sen tulosta, ja sen suoritus lopetetaan ohjelmasuorituksen g päättyessä. Tämä johtaa siihen, että ohjelmaa f ajetaan niin vähän kuin mahdollista. Tällaista tarvepohjaista suoritusta voidaan Hughesin (1989) mukaan käyttää sivuvaikutuksista johtuen vain funktionaalisessa ohjelmoinnissa. (Hughes, 1989)

## 3.3 Funktionaalisuus arviointiperusteena

Koska funktionaalinen ohjelmointi on “vain” paradigma, teoriassa funktionaalisia ohjelmia voitaisiin kirjoittaa millä tahansa ohjelmointikielellä, vaihtelevalla menestyksellä (Hu ym., 2015). Funktionaalisen ohjelmointikielen määritelmä ei olekaan yksiselitteinen (Fogus, 2013).

Ohjelmointikieltä kutsutaan usein funktionaaliseksi kieleksi, jos sen ominaisuudet kannustavat tai pakottavat käyttämään funktionaalista paradigmaa (Hu ym., 2015). Fogusin (2013) mukaan toisen yleisen määritelmän mukaan ohjelmointikieltä voi kutsua funktionaaliseksi kieleksi, jos se mahdollistaa ensiluokkaisten funktioiden luomisen ja käytön. Tyypillisesti määritelmään liitetään määrittelijästä riippuen rajoituksia. Esimerkiksi joidenkin mielestä funktionaalisen ohjelmointikielen täytyy sisältää staattista tyypitystä, hahmonsovitusta (pattern matching), muutumattomuutta, puhtautta, jne. Fogus (2013) määrittelee myös JavaScriptin funktionaaliseksi ohjelmointikieleksi koska funktiot ovat siinä ensiluokkaisia. (Fogus, 2013) Lisäksi Wadlerin (1995) mukaan funktionaaliset ohjelmointikielet voidaan jakaa kahteen ryhmään. Puhtaat kielet, kuten Miranda ja Haskell perustuvat suoraan lambdakalkyyliin. Epäpuhtaat kielet, kuten Scheme ja Standard ML laajentavat lambdakalkyyliä esimerkiksi sijoituslauseilla ja poikkeuksilla. Puhtaissa kielissä voidaan myös hyödyntää epäpuhtauden tuomia etuja *monadien* avulla. (Wadler, 1995)

Koska funktionaalisten ohjelmointikielten määritelmä ei ole yksiselitteinen, täytyy seuraavassa luvussa analysoitavien web-teknologioiden funktionaalisuutta arvioida funktionaalisen ohjelmointiparadigman ominaisuuksien avulla. Koska useat edellisissä luvuissa esitellyistä funktionaalisen ohjelmoinnin ominaisuuksista ovat suoraa seurausta jostakin toisesta tai useammasta ominaisuudesta, täytyy päättää ne funktionaaliset perusominaisuudet, jotka eivät ole seurausta jostakin, vaan johon muut ominaisuudet pohjautuvat. Edellisiin lukuihin perustuen näitä arviointikohteiksi soveltuvia perusominaisuuksia ovat muuttumattomuus, tilattomuus ja funktioiden ensiluokkaisuus.

# 4. Funktionaalisen ohjelmoinnin soveltuvuus web-käyttöliittymäkehitykseen

Viime vuosina funktionaalisia ohjelmointikieliä on alettu etujensa vuoksi hyödyntää useissa laajoissa web-sovellusprojekteissa. Esimerkiksi Facebook käyttää Haskell-kieltä uutisvirran luomisessa, WhatsApp Erlang-kieltä viestityspalvelimella, ja Twitter, LinkedIn ja Tumblr käyttävät Scala-kieltä järjestelmiensä infrastruktuurin ytimessä. (Hu ym., 2015)

Web-kehityksessä funktionaalisia ohjelmointikieliä on perinteisesti käytetty lähinnä palvelimella, esimerkiksi edellä mainituilla Erlang, Heskell, Scala ja lisäksi mm. Clojure -ohjelmointikielillä. Web-sovellusten kasvaneet vaatimukset ovat kuitenkin lisänneet tarvetta edistyneemmille ohjelmointikielille ja komponenteille myös web-käyttöliittymissä. (McGranaghan, 2011)

Mitä interaktiivisempi web-sovellus on, sitä enemmän kertyy yhteistä logiikkaa palvelimen ja käyttöliittymän välille, ja sitä kompleksisempaa käyttöliittymän ohjelmakoodi on (Richard-Foy ym., 2011). Eri teknologioiden yhdistelemisestä aiheutuvia ongelmia voidaan ratkaista kerroksettomalla ohjelmoinnilla (tierless programming), jossa useaa tai jokaista dynaamisen web-sovelluksen kerrosta ja komponenttia ohjataan yhdellä - mahdollisesti funktionaalisella - ohjelmointikielellä (Chlipala, 2016). Määrittelytavasta riippuen, yksi kerroksettoman ohjelmoinnin mahdollistava ohjelmointikieli on JavaScript (Philips, De Meuter & De Roover, 2015).

## 4.1 JavaScriptin funktionaaliset ominaisuudet ja laajennukset

Kuten aiemmin todettiin, funktionaalisen ohjelmointikielen määritelmä ei ole yksiselitteinen. JavaScript ei kaikkien määritelmien mukaan ole funktionaalinen ohjelmointikieli, mutta se on joustavimpia yleiskäyttöön suunnatuista ohjelmointikielistä. JavaScript mahdollistaakin erilaisten ohjelmointimenetelmien käytön, mutta ei pakota tiettyyn muottiin. JavaScriptissä on vain yhdenlainen funktio, ja käyttötavasta riippuen se voi toimia joko puhtaana funktiona, muuttuvana (mutable) proseduurina tai olion metodina. Funktio on yksi JavaScriptin ydinelementeistä. (Fogus, 2013)

```
[1, 2, 3].forEach(alert);
// alert box with “1” pops up
// alert box with “2” pops up
// alert box with “3” pops up
```

Yllä olevassa JavaScript-lähdekoodissa Array-luokan *forEach*-metodi saa parametrina funktion *alert*, ja syöttää sille taulukon elementit yksi kerrallaan. (Fogus, 2013)

Viime vuosina JavaScriptiin on *forEach*-metodin lisäksi lisätty tuki myös useille muille funktionaalisessa ohjelmoinnissa oleellisille korkeamman asteen funktioille, esimerkiksi Array-luokan *map*, *filter* ja *reduce*-metodeille. (Ecma International, 2011) Funktionaalisen ajattelutavan havainnollistamiseksi näitä metodeja esitellään seuraavassa.

map() suorittaa taulukon jokaiselle elementille parametrina annetun callback-funktion ja palauttaa uuden luodun taulukon:

```
var numbers = [1, 4, 9];
```

```
var roots = numbers.map(Math.sqrt);
// roots is now [1, 2, 3]
// numbers is still [1, 4, 9]
```

filter() palauttaa uuden taulukon jonka elementteinä ovat elementit jotka läpäisevät parametrina annetun callback-funktion testin:

```
function isBigEnough(value) {
  return value >= 10;
}
var filtered = [12, 5, 8, 130, 44].filter(isBigEnough);
// filtered is [12, 130, 44]
```

reduce() suorittaa taulukon elementeille callback-funktion jonka parametreistä ensimmäinen on edeltävä palautettu arvo (tai initialValue, esimerkissä 0) ja jälkimmäinen on nykyinen käsiteltävä arvo.

```
var sum = [0, 1, 2, 3].reduce(function(a, b) {
  return a + b;
}, 0);
// sum is 6
```

(Mozilla Developer Network, 2016)

JavaScript siis mahdollistaa monien funktionaalisen ohjelmoinnin ominaisuuksien hyödyntämisen mutta ei pakota niiden käyttöä. JavaScriptillä ei voida määritellä muuttujaa jonka arvo ei voi muuttua, mutta se tarjoaa metodeja, joiden avulla muuttujia voidaan käsitellä niin kuin ne olisivat muuttumattomia. Jos muuttumattomuus halutaan varmistaa, joudutaan käyttämään jotain apukirjastoa, kuten Immutable.js (Immutable.js, 2016). Vastaavasti JavaScriptillä voidaan toteuttaa tilaa muuttamattomia funktioita, mutta monimutkaisten käyttöliittymien toteutus yleensä myös monimutkaista tilan hallintaa. Tilan hallintaa voidaan helpottaa esimerkiksi Redux-kirjaston avulla (Redux, 2016). JavaScriptissä funktiot ovat ensiluokkaisia, mikä mahdollistaa korkeamman asteen funktiot. Myös suorituksen rinnakkaistaminen on mahdollista nykyaikaisissa selaimissa HTML5:n Web Workers-API:n avulla (Mozilla Developer Network, 2016).

Underscore.js on JavaScriptin funktionaalisia ominaisuuksia laajentava kirjasto. JavaScriptin omia tietorakenteita ja olioita se ei laajenna, vaan tarjoaa laajentavat ominaisuudet omana kokonaisuutenaan. Underscore määrittelee nimensä mukaisen globaalin ’_’ olion, joka sisältää hyödyllisiä funktionaalista ohjelmointia helpottavia metodeja. Fogus:in (2013) esimerkissä ’_’ oliota käytetään seuraavasti:

```
_.times(4, function() { console.log("Major") });
```

Tämä funktionaalista ajattelutapaa noudattava esimerkki tulostaa konsoliin 4 kertaa tekstin ”Major”. (Fogus, 2013)

React.js on Facebook Inc.:n vuonna 2013 julkaisema käyttöliittymien rakentamiseen tarkoitettu JavaScript-kirjasto. React on deklaratiivinen API, jonka periaatteena on mahdollistaa imperatiivisen API:n käyttäminen deklaratiivisesti. React siis kannustaa funktionaaliseen ajattelutapaan käyttöliittymäkehityksessä. (Fedosejev, 2015)

React ratkaisee aiemmin mainitun DOM-manipulaation imperatiivisuus- ja suorituskykyongelman virtuaalisen DOM-ratkaisun (Virtual DOM) avulla. Virtuaalinen DOM on nopea, keskusmuistissa sijaitseva abstraktio, joka kuvaa oikeaa DOM-puuta. Kun tietomallin tila muuttuu, React ja virtuaalinen DOM luovat uuden käyttöliittymän

virtuaalisen DOM-esityksen. React vertailee edellistä virtuaalisen DOM:n tilaa uuteen tilaan. Vertailun tulokseksi laskettu muutos on se, jonka React päivittää oikeaan DOM-rakenteeseen. (Fedosejev, 2015)

Virtual DOM-menetelmän etuna on, että tilan muuttuessa selaimessa käytettävää DOM-puuta manipuloidaan vain niiltä osin kuin on tarpeellista. Merkittävä etu on myös se, että sovellusta kehitettäessä Reactilla, kehittäjän ei tarvitse huolehtia siitä, mitä DOM:iin todellisuudessa päivitetään, vaan sovellusta voidaan kehittää niin kuin koko DOM päivittyisi aina tilan muuttuessa. (Fedosejev, 2015)

Seuraava Fedosejevin (2015) funktionaalista ajattelutapaa noudattava esimerkki kuvaa h1-tyyppisen DOM-elementin luomista Reactilla:

```
var React = require('react');
var ReactDOM = require('react-dom');
var reactElement = React.createElement('h1', { className: 'header' });
ReactDom.render(reactElement, document.getElementById('application'));
```

Ohjelma ottaa käyttöön tarvittavat React-komponentit ja luo niiden avulla h1-elementin, joka sijoitetaan "application"-tunnisteella löydettävään DOM-elementtiin.

## 4.2 JavaScriptiksi käännettävät ohjelmointikielet

Suoran JavaScript-ohjelmoinnin sijaan web-sovelluksia voidaan ohjelmoida kyseiseen käyttöön tarkoitetulla kielellä, joka käännetään kääntäjän avustuksella JavaScriptiksi ennen käyttöönottoa. Esimerkkejä käännettävistä kielistä ovat mm. TypeScript ja seuraavassa esiteltävät funktionaaliseen ohjelmointiin tarkoitetut ClojureScript ja Elm.

### 4.2.1 ClojureScript

Lisp-kieleen pohjautuva funktionaalinen ohjelmointikieli Clojure kehitettiin alun perin Java Virtual Machine (JVM) -alustalle. ClojureScript mahdollistaa Clojuren ominaisuuksien käyttämisen web-käyttöliittymäkehityksessä ClojureScript-JavaScript -kääntäjän avulla. Vaikka Clojure ja ClojureScript on kehitetty eri alustoja ja käyttötarkoituksia varten, kielten ydintoiminnallisuudet ovat samanlaiset. (McGranaghan, 2011)

Vaikka ClojureScript kääntyy JavaScriptiksi, sen kääntäjän ja ajonaikaisen kirjaston (runtime library) avulla voidaan ottaa käyttöön funktionaalisten kielten ominaisuuksia joita JavaScriptillä ei voisi samalla tavalla toteuttaa. JavaScriptissä itsessään on vain yksi tietorakenne, mutta ClojureScript tuo mukanaan valikoiman funktionaalisia muuttumattomia ja versioituvia (persistent) tietorakenteita. Toisin kuin JavaScriptin ylle rakentuvat syntaktiset kerrokset kuten CoffeeScript, ClojureScript on semantiikkaa muuttava kääntäjä. (McGranaghan, 2011)

Koska ClojureScript noudattaa funktionaalista paradigmaa sen kaikkien periaatteiden mukaisesti, se täyttää jokaisen aiemmin määrittelemämme arviointikohteen vaatimukset.

### 4.2.2 Elm

Elm on Evan Czaplickin kehittämä syntaksiltaan Haskell-kieltä muistuttava puhtaan funktionaalinen ohjelmointikieli, joka poikkeaa muista käännettävistä kielistä siten, että JavaScriptin lisäksi se kääntyy myös HTML:ksi ja CSS:ksi. Elm soveltuu siis aiemmin mainittuun kerroksettomaan ohjelmointimenetelmään. Elm-kielessä funktionaalisen periaatteen mukaisesti muuttuja ei voi vaihtaa arvoaan arvon saatuaan. Elm käyttää aiemmin kuvattua virtuaalista DOM-tekniikkaa sovelluksen renderöintiin. (Czaplicki, 2016; Eriksson & Ärleryd, 2016)

Jokainen Elm-ohjelma perustuu logiikkaan, joka voidaan jakaa kolmeen selvästi toisistaan erottuvaan osaan. Malli (model) sisältää sovelluksen tilan, päivitys (update) on tapa päivittää tila, ja näkymä (view) on tapa näyttää tila HTML-muodossa. Czaplicki (2016) kuvaa logiikkaa seuraavan esimerkin avulla:

```
import Html exposing (Html, button, div, text)
import Html.Events exposing (onClick)

main =
  Html.beginnerProgram { model = model, view = view, update = update }

-- MODEL

type alias Model = Int

model : Model
model =
  0

-- UPDATE

type Msg = Increment | Decrement

update : Msg -> Model -> Model
update msg model =
  case msg of
    Increment ->
      model + 1

    Decrement ->
      model - 1

-- VIEW

view : Model -> Html Msg
view model =
  div []
    [ button [ onClick Decrement ] [ text "-" ]
    , div [] [ text (toString model) ]
    , button [ onClick Increment ] [ text "+" ]
    ]
```

Ohjelma tulostaa numeron (aluksi 0) ja kaksi painiketta jolla tulostettavan numeron arvoa voidaan kasvattaa tai vähentää. (Czaplicki, 2016)

Myös Elm noudattaa funktionaalista paradigmaa sen kaikkien periaatteiden mukaisesti, joten se täyttää jokaisen aiemmin määrittelemämme arviointikohteen vaatimukset.

# 5. Pohdinta

Kuvausten (mm. Hughes, 1989; Hudak & Bloss, 1985; Fogus, 2013) perusteella funktionaalinen ohjelmointi on erinomainen ohjelmiston ylläpitoa, virheenkorjausta ja rinnakkaissuoritusta helpottava ohjelmointiparadigma. Sivuvaikutuksien puuttumisesta periytyvät edut ovat selvästi havaittavissa, mutta funktionaalinen ohjelmointi tuo myös mukanaan uusia ongelmia. Kuten Knight (1986) totesi, kokemattomalle ohjelmoijalle funktionaalinen ajattelutapa on vaikeasti hahmotettavissa. Imperatiivisessa ohjelmoinnissa luonteenomainen algoritmin perustuminen tilasiirtymien ketjuun on usein helpompaa suunnitella, ja esimerkiksi rekursiivisten ja korkeamman asteen funktioiden suunnittelu vaatii ohjelmoijalta kykyä lähestyä ongelmaa täysin erilaisesta näkökulmasta.

Tutkimuksen tarkoituksena oli selvittää, mitä funktionaalisen ohjelmointiparadigman tärkeimpiä ominaisuuksia voidaan hyödyntää web-käyttöliittymäkehityksessä, ja millä tavalla niitä voidaan hyödyntää. Fogus (2013) mainitsi, että lisääntyvän interaktiivisuuden vaatiman DOM-manipulaation mahdollistava JavaScript ei kaikkien määritelmien mukaan ole funktionaalinen ohjelmointikieli. Havaintojen mukaan JavaScript kuitenkin sisältää funktionaalisen ohjelmoinnin mahdollistavia ominaisuuksia, mutta ei pakota tai erityisemmin kannustakaan niiden ominaisuuksien hyödyntämistä.

Tutkimuskysymykseen voidaan siis vastata, että kaikkia funktionaalisen ohjelmointiparadigman tärkeimpiä ominaisuuksia voidaan hyödyntää web-käyttöliittymäkehityksessä, tietyssä määrin pelkällä JavaScriptillä, mutta on havaittavissa tarvetta laajentaa JavaScriptin funktionaalisia ominaisuuksia esimerkiksi muuttumattomuuden (Immutable.js, 2016) ja tilan hallinnan (Redux, 2016) avulla jotta funktionaalista ohjelmointiparadigmaa voidaan hyödyntää paremmin. Esimerkiksi, koska JavaScriptin muuttujat eivät ole muuttumattomia, ei voida olla varmoja funktioiden ja lausekkeiden sivuvaikutuksettomuudesta. Lisäksi esitellyn Underscore.js:n apumetodien ja React.js:n virtuaalisen DOM-ratkaisun tarjoamat edut ovat merkittäviä (Fogus, 2013; Fedosejev, 2015). JavaScriptin funktionaalisten ominaisuuksien laajentamisen lisäksi toinen havaittu funktionaalisen paradigman hyödyntämisen mahdollistava vaihtoehto on käyttää funktionaaliseen ohjelmointiin tarkoitettua ohjelmointikieltä, kuten ClojureScript tai Elm, jotka käännetään JavaScriptiksi.

Mainituista syistä, ja esimerkiksi Cervesaton ja Sansyzin (2014) mukaan, nykyaikaisten web-sovellusten kehittäjän tulee hallita laajaa valikoimaa uusia ja heterogeenisiä teknologioita ja menetelmiä. Eri kieliin ja paradigmoihin perehtyminen on silti usein hyödyllistä. Esimerkiksi funktionaalista ohjelmointia pystyy soveltamaan tietyllä tasolla imperatiiviseen ohjelmointiin. Myöskään ohjelmointikielten väliset erot eivät usein ole suuria (esim. Flanagan, 2006), joten yhden kielen osaaminen auttaa oppimaan toisen kielen nopeammin.

# 6. Yhteenveto

Web-sovellusten vaatimukset ovat kasvaneet ja siitä johtuen web-käyttöliittymien kehitys on monimutkaistunut. Funktionaalisen ohjelmointiparadigman tarjoamat edut ohjelmistokehityksessä ovat esiteltyjen tutkimuksien ja kirjallisuuden mukaan selvästi havaittavissa. Näistä syistä aiemmin esitetty tutkimuskysymys, eli mitä funktionaalisen ohjelmointiparadigman tärkeimpiä ominaisuuksia voidaan hyödyntää web-käyttöliittymäkehityksessä, ja millä tavalla niitä voidaan hyödyntää, osoittautui tärkeäksi.

Tutkimuksessa havaittiin, että kaikki funktionaalisen ohjelmointiparadigman tärkeimmät ominaisuudet ovat hyödynnettävissä web-käyttöliittymäkehityksessä. Siihen, miten niitä voidaan hyödyntää, löydettiin kaksi ratkaisua. JavaScriptin ja sille kehitettyjen kirjastojen, kuten Immutable.js, Redux, Underscore.js ja React, avulla voidaan näitä ominaisuuksia hyödyntää niissä määrin kuin nähdään tarpeelliseksi. Toinen ratkaisu on käyttää erityisesti funktionaaliseen ohjelmointiin suunnattuja käännettäviä ohjelmointikieliä, kuten ClojureScript ja Elm.

Seurauksena näistä ratkaisuista on interaktiivisten web-sovellusten kehityksessä vähentyneet sivuvaikutuksien aiheuttamat ongelmat. Sovelluskehittäjälle ratkaisusta seuraa lisääntynyt tarve perehtyä uusiin ohjelmointimenetelmiin ja -kieliin.

Vie oman aikansa, että tieteellinen tutkimus kykenee arvioimaan uusia teknologioita. Lisätutkimuksen tarvetta on havaittavissa useiden funktionaalisuuteen pohjautuvien teknologioiden soveltamisessa web-kehitykseen. Esimerkiksi viimeisessä luvussa mainittu Elm-kieli on hyvin tuore, ja siitä on toistaiseksi vaikea löytää kattavaa tutkimusta. Tässä tutkielmassa ei otettu kantaa siihen, kuinka tehokkaita tarjotut ratkaisut ovat. Tarvetta lisätutkimukselle on siis mainittujen ratkaisujen soveltuvuudessa erityyppisiin ohjelmistokehitysprojekteihin ja niiden tehokkuuden mittaamiseen kyseisessä kontekstissa.

# Lähteet


Achten, P., & Koopman, P. (2013). *The beauty of functional code: Essays dedicated to rinus plasmeijer on the occasion of his 61st birthday* Springer.

Backhouse, R., Jansson, P., Jeuring, J., & Meertens, L. (1998). Generic programming. *International School on Advanced Functional Programming,* pp. 28-115.

Cervesato, I., & Sansyz, T. (2014). Substructural meta-theory of a type-safe language for web programming. *Fundamenta Informaticae, 130*(1), 67-97.

Chlipala, A. (2016). Ur/Web: A simple model for programming the web. *Communications of the ACM, 59*(8), 93-100.

De Bakker, J. W., & De Vink, E. (1996). *Control flow semantics* Citeseer.

Ecma International. (2011). *Final draft standard ECMA-262 edition 5.1.* Viitattu 30.12.2016. Saatavilla: http://www.ecma-international.org/ecma-262/5.1/Ecma-262.pdf

Eriksson, N., & Ärleryd, C. (2016). Elmulating JavaScript.

Evan Czaplicki. (2016). *An introduction to elm.* Viitattu 30.12.2016. Saatavilla: https://guide.elm-lang.org/

Fedosejev, A. (2015). *React. js essentials* Packt Publishing Ltd.

Flanagan, D. (2006). *JavaScript: The definitive guide* " O'Reilly Media, Inc.".

Fogus, M. (2013). *Functional javascript*. Sebastopol, CA: O'Reilly Media, Inc.

Garrett, J. J. (2005). Ajax: A new approach to web applications.

Ginige, A., & Murugesan, S. (2001). Web engineering: An introduction. *IEEE Multimedia, 8*(1), 14-18.

Hu, Z., Hughes, J., & Wang, M. (2015). How functional programming mattered. *National Science Review, 2*(3), 349-370.

Hudak, P., & Bloss, A. (1985). The aggregate update problem in functional programming systems. *Proceedings of the 12th ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages,* pp. 300-314.

Hughes, J. (1989). Why functional programming matters. *Computer Journal, 32*(2), 98-107.

Immutable.js. (2016). *Immutable.js.* Viitattu 29.12.2016. Saatavilla: https://facebook.github.io/immutable-js/

Knight, T. (1986). An architecture for mostly functional languages. *Proceedings of the 1986 ACM Conference on LISP and Functional Programming,* pp. 105-112.

McGranaghan, M. (2011). ClojureScript: Functional programming for JavaScript platforms. *IEEE Internet Computing, 15*(6), 97-102.

Mozilla Developer Network. (2016). *Web technology for developers.* Viitattu 10.12.2016. Saatavilla: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array ja https://developer.mozilla.org/en-US/docs/Web/API/Web_Workers_API

Negrino, T., Smith, D., & Kamppila, M. (2007). *JavaScript : Tehokas hallinta*. Helsinki: Readme.fi.

Philips, L., De Meuter, W., & De Roover, C. (2015). Poster: Tierless programming in JavaScript. *2015 IEEE/ACM 37th IEEE International Conference on Software Engineering, , 2.* pp. 831-832.

Pressman, R. S. (2010). *Software engineering : A practitioner's approach* (7th ed ed.). Boston (MA): McGraw-Hill.

Redux. (2016). *Redux documentation.* Viitattu 29.12.2016. Saatavilla: http://redux.js.org/docs/introduction/

Richard-Foy, J., Barais, O., & Jezequel, J. (2014). Efficient high-level abstractions for web programming. *Acm Sigplan Notices, 49*(3), 53-60.

Robbins, J. N. (2012). *Learning web design: A beginner's guide to HTML, CSS, JavaScript, and web graphics* " O'Reilly Media, Inc.".

Vesterholm, M., & Kyppö, J. (2015). *Java-ohjelmointi* (9. uud. p. ed.). Helsinki: Talentum.

Wadler, P. (1995). Monads for functional programming. *International School on Advanced Functional Programming,* pp. 24-52.

World Wide Web Consortium. (2016). *Html & css.* Viitattu 10.12.2016. Saatavilla: https://www.w3.org/standards/webdesign/htmlcss